1 What is an algorithm?

An algorithm is a sequence of unambiguous instructions for solving a problem, that is, for obtaining a required output for any legitimate input in a finite amount of time.

\[
\text{instance} \rightarrow \text{algorithm} \rightarrow \text{output}
\]

Analysis of algorithms is the quantitative study of the performance of algorithms, in terms of their run time, memory usage, or other properties.

2 What is this course about?

Most of the course will blend two parallel goals:

- **Techniques** for analyzing algorithms.
- **Applications** of those techniques to important algorithms and data structures.

3 Models of computation

We can make the idea of sequence of instructions precise by defining a model of computation.

One important early model of computation is the Turing machine which includes:

- A finite, non-empty set of states \( Q \).
- A finite, non-empty set of tape symbols \( \Gamma \).
- A blank symbol \( b \in \Gamma \).
- A finite set of input symbols \( \Sigma \).
- A transition function \( \delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\} \),
- An initial state \( q_0 \in Q \) and a set of final states \( F \subseteq Q \).

Informally, we can think of a Turing machine as a finite state machine that reads and writes from an infinitely-long strip of tape. The main idea here is that, though the Turing machine model is very powerful and expressive, it is also cumbersome to use — essentially no one describes algorithms in it directly except in college classes on the theory of computation.
4 RAM model
Another, more manageable option:

Random-access machine (RAM) model (informal summary)

- Simple operations (arithmetic, comparison, conditional, etc.) each take the same, constant amount of time.
- Data stored in an infinite array of registers (0, 1, 2, …), each of which can hold $c \log n$ bits.
  - $n$ – problem size
  - $c$ – some constant independent of $n$

In most cases, this level of detail is unnecessary for understanding how an algorithm works. However, it’s important to have a formal model behind the scenes; without this, it’s meaningless to try to prove anything about an algorithm or its performance.

5 Example: Sorting
Sorting is a problem that is practically important and useful for illustrating many recurring ideas in algorithms.

- **Input:** A sequence of numbers $\langle a_1, \ldots, a_n \rangle$.
- **Output:** A reordering of those numbers, denoted $\langle a'_1, \ldots, a'_n \rangle$, such that
  \[ a'_1 \leq a'_2 \leq \cdots \leq a'_n. \]

Note that the idea of “sorting” is not restricted to just numbers. As long as the elements are drawn from a totally ordered set, then the problem is still well defined. We’ll use numbers through this course because they make the intuition very easy.

6 Example: Insertion sort

```
IN SERT ION S ORT(A)
    for j = 2, …, A.length do
        k = A[j]
        i = j - 1
        while i > 0 and A[i] > k do
            A[i + 1] = A[i]
            i = i - 1
        end while
        A[i + 1] = k
    end for
```
7 Example: Mergesort

\[
\text{MERGESORT}(A, \ell, r)
\]
\[
\text{if } \ell < r \text{ then}
\]
\[
m = \lfloor(\ell + r)/2\rfloor
\]
\[
\text{MERGESORT}(A, \ell, m)
\]
\[
\text{MERGESORT}(A, m + 1, r)
\]
\[
\text{MERGE}(A, \ell, m, r) \quad \text{// CLRS pg 31}
\]
\[
\text{end if}
\]